CLAIMS

We claim:

1. A computer readable medium storing computer executable instructions for performing a method of transforming a programming language specification into a lower-level specification, the method comprising:

accepting a programming language specification for a design unit, the programming language specification including a public interface; and

based upon a set of transformation rules, transforming plural methods of the public interface into plural ports of a port map of a lower-level specification for the design unit.

2. The computer readable medium of claim 1 wherein the method further comprises:

based upon the set of transformation rules, transforming an algorithmic method implementation of the programming language specification into a synchronized process of the lower-level specification.

3. The computer readable medium of claim 1 wherein the transforming the public interface includes:

transforming each of zero or more input methods into an input port of the port map; and

transforming each of zero or more output methods into an output port of the port map.

4. The computer readable medium of claim 1 wherein a first input method of the public interface includes as a parameter a pointer to a shared variable, wherein the shared variable is for modeling inout behavior.

5. The computer readable medium of claim 1 wherein the transforming the public interface further includes:

based upon the set of transformation rules, transforming plural native programming language data types into hardware description language data types.

6. The computer readable medium of claim 1 wherein the programming language specification includes one or more values specified in a template, the programming language specification based upon the template.

7. The computer readable medium of claim 1 wherein the programming language specification is part of a system including hardware and software.

8. The computer readable medium of claim 1 wherein the programming language specification is a C++ class description.

9. A computer readable medium storing computer executable instructions for performing a method of transforming a programming language specification into a lower-level specification, the method comprising:

accepting a programming language specification for a design unit, the programming language specification including plural calls to plural instances of a sub-design unit class, wherein a first call of the plural calls maps to a first instance of the sub-design unit class, and wherein a second call of the plural calls maps to a second instance of the sub-design unit class; and

transforming the programming language specification into a lower-level specification for the design unit, wherein the transforming includes generating lower-level description for handling concurrent execution of sub-design units represented by the plural instances of the sub-design unit class in the programming language specification.

10. The computer readable medium of claim 9 wherein the programming language specification also lacks synchronization modeling, and wherein the

transforming further includes generating lower-level description to model synchronization for the design unit.

11. The computer readable medium of claim 9 wherein the method further comprises:

transforming an algorithmic method implementation of the programming language specification into a process of the lower-level specification.

12. The computer readable medium of claim 9 wherein the method further comprises:

transforming plural methods of a public interface of the programming language specification into plural ports of a port map of the lower-level specification.

13. The computer readable medium of claim 9 wherein the programming language specification is an object-oriented class description.

14. A design tool comprising:

a design input module for accepting an algorithmic specification of a design unit, the algorithmic specification including plural sub-design unit calls that map to plural different instances of a sub-design unit, thereby indicating parallel execution of the plural sub-design unit calls; and

a hardware description language transformer for transforming the algorithmic specification into a lower-level specification, wherein the transformer adds code into the lower-level specification for handling the parallel execution of the plural sub-design unit calls.

15. The design tool of claim 14 further comprising:

an architecture exploration module for presenting alternative architectures for the lower-level specification to a designer.

16. The design tool of claim 14 wherein instantiation relationships in the algorithmic specification represent structural relationships in the design unit.

17. A design tool comprising:

one or more modules for accepting an object-oriented description for a design unit, the object-oriented description including a public interface; and

one or more modules for translating the object-oriented description into a lower-level specification according to defined semantics for the public interface.

18. The design tool of claim 17 wherein the public interface includes zero or more input methods, zero or more output methods, and an algorithmic method.

19. The design tool of claim 17 wherein the object-oriented description further comprises a private interface comprising one or more declarations, the declarations for internal signals used in methods of the public interface, and wherein the translating comprises:

translating each of zero or more input method calls into an assignment to a corresponding input signal;

generating a synchronization scheme for an algorithmic method; and

translating each of zero or more output method calls into a read of a corresponding output signal.

20. The design tool of claim 17 further comprising:

one or more modules for presenting alternative architectures for the design unit to a designer.

21. In a computer system, a method comprising:

receiving an object-oriented description of a design unit, the object-oriented description including native programming language code, wherein the object-oriented description specifies structural details of the design unit; and

compiling the object-oriented description with a native programming language compiler.

22. The method of claim 21 wherein the native programming language is C++, and wherein the object-oriented description follows C++ syntax.

23. The method of claim 21 further comprising:

performing algorithmic simulation with an executable produced by the compiling.

24. A computer readable medium storing an algorithmic specification for a design unit, the specification including:

a public interface comprising:

for each input port of a design unit, an input method;

an algorithmic method; and

for each output port of the design unit, an output method.

25. The computer readable medium of claim 24 wherein the algorithmic specification further comprises a class declaration annotating the algorithmic specification as a structural unit.

26. The computer readable medium of claim 24 wherein each input method has one input value and no return value, wherein each output method has no input value and one return value, and wherein the algorithmic method has no input value and no return value.

27. The computer readable medium of claim 24 wherein a first input method includes as a parameter a pointer to a shared variable.

28. The computer readable medium of claim 24 wherein the algorithmic specification further comprises a constructor for a reset action.

29. The computer readable medium of claim 24 wherein one or more values of the algorithmic specification are specifiable in a template for the algorithmic specification.

30. The algorithmic specification of claim 24 wherein the public interface includes only native constructs, the algorithmic specification compilable by a native programming language compiler for algorithmic simulation.

31. In a computer system, a method of modeling caller behavior of a design unit with a software object, the method comprising:

calling any input methods of a software object, each call to an input method corresponding to a setting a value of a port of a design unit;

calling an algorithmic method of the software object, the algorithmic method corresponding to processing of the design unit; and

calling any output methods of the software object, each call to an output method corresponding to a retrieving a value of a port of the design unit.

32. In a design tool, a method for implementing a hierarchical relationship between a first design unit and a second design unit, the method comprising:

automatically creating one or more local signals of a first design unit, the first design unit including an instance of a second design unit; and

mapping the one or more local signals to corresponding ports of the second design unit.

33. The method of claim 32 further comprising:

before the creating, accepting an algorithmic specification of the first design unit.

34. The method of claim 32 further comprising:

outputting a hardware description language specification for the first design unit.

35. The method of claim 32 wherein the mapping comprises:

for each input method call to a public interface for the second design unit, translating the input method call into an assignment to a corresponding port signal for the second design unit; and

for each output method call to the public interface, translating the output method call into a read of a corresponding output port signal for the second design unit.

36. A method of testing a design unit, the method comprising:

simulating algorithmic behavior of a design unit based upon execution of a programming language specification of the design unit; and

simulating waveform input to the design unit based upon execution of the programming language specification within a wrapper object, the wrapper object interfacing a hardware description language simulator.

37. The method of claim 36 further comprising:

transforming the programming language specification into a hardware description language specification; and

simulating hardware behavior of the design unit based upon execution of the hardware description language specification.

38. The method of claim 36 further comprising:

simulating behavior of the design unit operating with one or more other design units.

39. A method of creating a system including software and hardware, the method comprising:

accepting a programming language specification of a system including software and hardware, wherein the programming language specification follows the same syntax for the software and the hardware; and

transforming at least part of the programming language specification into a lower-level specification.

40. The method of claim 39 further comprising:

simulating the transformed lower-level specification along with any non-transformed portions of the programming language specification, wherein the simulating co-verifies the software and hardware of the system.